Here’s a clean, opinionated design you can hand to an architect / team and start building.

1. High-level architecture

Core pieces:

1. MongoDB Cluster
    - 3+ node replica set (or Atlas cluster).
    - One logical database: plansdb.
2. Plan Data SDK / Library
    - Internal library that owns all Mongo access.
    - Used by:
        - Plan API Service (external + internal APIs).
        - Nightly Plan Updater Job.
        - Any other internal jobs needing DB access.
3. Plan API Service
    - Stateless microservice (REST or GraphQL).
    - Exposes plan/rate data to internal & external consumers.
    - Talks to Mongo only via the SDK.
4. Nightly Updater Job
    - Scheduled job (cron, Airflow, Kubernetes CronJob, etc.).
    - Fetches source plan data (file feed, SFTP, external API, etc.).
    - Uses SDK for all DB writes (bulk upserts).
5. API Gateway / Edge / WAF
    - Exposes the external Plan API.

- Enforces authn/authz (OAuth2/OIDC), rate limits, WAF rules.

6. Observability Stack
    - Centralized logging, metrics, tracing.
    - Mongo monitoring (Atlas or Ops Manager + dashboards).

2. Data model in MongoDB

Assume 25k plans per year. That’s not huge, but we’ll design properly so it scales.

2.1 Collections

plans collection (immutable-ish plan metadata)

```
{
  "_id": ObjectId(),
  "planId": "CARRIER123-2026-PLN-001",
  "carrierId": "CARRIER123",
  "year": 2026,
  "name": "Gold PPO 2000",
  "productType": "PPO",
  "metalTier": "GOLD",
  "networkId": "NETWORK_ABC",
  "effectiveFrom": ISODate("2026-01-01T00:00:00Z"),
  "effectiveTo": ISODate("2026-12-31T23:59:59Z"),
  "status": "ACTIVE",        // ACTIVE | INACTIVE | DRAFT
  "serviceArea": {
    "country": "US",
    "state": "GA",
    "ratingArea": "GA-01",
    "countyFips": ["13117", "13089"]
  },
  "attributes": {            // flexible key-value payload
    "deductibleIndividual": 2000,
    "deductibleFamily": 4000,
    "oopMaxIndividual": 7000,
    "oopMaxFamily": 14000,
    "hsaEligible": true
  },
  "tenantId": "TENANT_01",   // if multi-tenant
```

```
  "version": 5,              // internal versioning
  "createdAt": ISODate(),
  "updatedAt": ISODate(),
  "isDeleted": false
}
```

Indexes:

- planId (unique)
- { year: 1, carrierId: 1, planId: 1 }
- { tenantId: 1, year: 1 } if multi-tenant
- Optional: { status: 1, effectiveFrom: 1, effectiveTo: 1 }

rates collection (rates for each plan, separated for flexibility)

```
{
  "_id": ObjectId(),
  "planId": "CARRIER123-2026-PLN-001",
  "year": 2026,
  "rateKey": {
    "state": "GA",
    "ratingArea": "GA-01",
    "age": 40,
    "coverageLevel": "INDIVIDUAL", // INDIVIDUAL | FAMILY
    "tobacco": false
  },
  "monthlyPremium": 410.55,
  "tenantId": "TENANT_01",
  "version": 5,
  "effectiveFrom": ISODate("2026-01-01T00:00:00Z"),
  "effectiveTo": ISODate("2026-12-31T23:59:59Z"),
  "createdAt": ISODate(),
  "updatedAt": ISODate(),
  "isDeleted": false
}
```

Indexes:

- { planId: 1, year: 1 }

- { year: 1, rateKey.state: 1, rateKey.ratingArea: 1, rateKey.age: 1 }
- { tenantId: 1, year: 1 }

2.2 Versioning strategy

To support nightly updates safely:

- Version field (version): Monotonically increasing per load (e.g. nightly job).
- Nightly job:
    1. Compute new version (e.g. N+1).
    2. Upsert all plans & rates with version=N+1.
    3. After successful load, mark older versions as isDeleted = true or just treat version = latest as “current”.
- Queries default to latest version per year + tenant (enforced by SDK).

3. Plan Data SDK / Library

Goal: “Core DB management done by a sdk/library to save data to DB and avoid too many http calls.”

3.1 Responsibilities

1. Mongo Connection Management
    - Connection pooling, retry policy, timeouts.
    - Reads/writes use proper read preference and write concern.
2. Data Access Layer (Repository)
    - PlanRepository and RateRepository with methods like:

        - getPlanById(planId, options?)
        - searchPlans(filters, pagination)
        - getRates(planId, filters)
        - bulkUpsertPlans(plans, version)
        - bulkUpsertRates(rates, version)
    - All queries automatically apply:
        - tenantId (if multi-tenant).
        - version = latest and isDeleted = false for “current” queries.
3. Validation & Schema
    - JSON schema validation for incoming plan/rate objects.
    - Reusable validation for both API and nightly job.
4. Caching & Batching (to reduce HTTP / DB traffic)
    - Optional in-memory cache (e.g. LRU) for very hot reads:
        - Keyed by (tenantId, planId, version).
        - TTL-based or version-based invalidation (on new nightly load).
    - Bulk methods:
        - getPlansByIds(planIds: string[]) to minimize round trips.
        - getRatesByPlanIds(planIds: string[]).
5. Telemetry
    - The SDK emits:
        - Counters: db_queries_total, cache_hits_total, bulk_upserts_total.
        - Timers: db_query_latency_ms.
        - Errors: db_errors_total.
6. Security Guardrails

    - Prevents collections from being used without tenant filter (if multi-tenant).
    - Central place for field-level encryption / decryption logic.

3.2 Where SDK is used

- Plan API Service
    - All plan/rate queries go through SDK.
- Nightly Updater Job
    - Uses bulkUpsertPlans and bulkUpsertRates.
    - No direct Mongo driver usage anywhere else.

4. Plan API Service

4.1 External API surface (examples)

REST example:

- GET /v1/plans
    - Query params: year, carrierId, state, ratingArea, metalTier, status, page, pageSize.
    - Uses SDK searchPlans.
- GET /v1/plans/{planId}
    - Returns plan + (optionally) rates.
    - Uses SDK getPlanById + getRates.
- GET /v1/plans/{planId}/rates

    - Query: state, ratingArea, age, coverageLevel, tobacco.
    - Uses SDK getRates.
- GET /v1/metadata/years or carriers
    - For UI dropdowns.

Internal-only endpoints:

- POST /internal/v1/plans/reindex (trigger background reindex).
- GET /internal/v1/health (health and readiness data).

4.2 Security

- AuthN:
    - External: OAuth2/OIDC (e.g., Azure AD, Auth0).
    - Internal: mTLS & JWT service tokens.
- AuthZ:
    - Claims in JWT: tenantId, roles, permissions.
    - Request scoped: the API applies tenantId filter for data isolation.
    - Roles: PLAN_READ, PLAN_ADMIN, etc.
- Input Validation & Sanitization
    - All user inputs validated before passed to SDK.
    - Pagination enforced (e.g. pageSize max 100 or 500).
- Network security
    - API behind Gateway + WAF.

    - Mongo not publicly reachable (VPC/private subnet, security groups, VNet peering).
    - Secrets stored in secret manager (Vault, AWS Secrets Manager, etc.).
- Data security
    - Mongo encryption at rest.
    - TLS between services and Mongo.
    - Optional field-level encryption for sensitive fields if any (e.g. negotiation prices, etc.).

5. Nightly Job Design

5.1 Flow

1. Trigger
    - Cron / Kubernetes CronJob or external scheduler (Airflow, etc.).
2. Ingestion
    - Fetch source data (SFTP, blob storage, external API).
    - Normalize into internal plan & rate JSON structures.
3. Validation
    - Apply same JSON schema as API (via SDK).
    - Log and quarantine invalid rows.
4. Versioning
    - Determine new version (e.g. version = previous_version + 1).
    - Store versioning info in a small metadata collection:

```
{
  "_id": "plans_version_tenant01_2026",
  "tenantId": "TENANT_01",
  "year": 2026,
  "currentVersion": 5,
  "previousVersions": [1,2,3,4],
  "updatedAt": ISODate()
}
```

4.

5. Bulk Upsert
    - Chunk data (e.g. batches of 1–5k docs).
    - Use SDK bulkUpsertPlans and bulkUpsertRates, using

      updateOne({planId, tenantId, year}, {$set: {...}}, {upsert: true}).
    - Make writes idempotent; re-running job overwrites same version.
6. Commit New Version
    - After successful upsert:
        - Update metadata.currentVersion.
        - Optionally mark older versions as isDeleted=true for read queries.
7. Emit Metrics & Alerts
    - Records loaded, failed, duration, size.
    - Integration with alerting if:
        - Job fails.
        - Loaded count deviates hugely from historical average.

6. Performance Considerations

Even though 25k/year is small, design to scale.

1. Indexes tuned to access patterns
    - Plan lookups by planId, year, carrierId, state, ratingArea, etc.
    - Rate lookups by (planId, year) and filters for state, ratingArea, age.
    - Periodically review and optimize indexes by analyzing query patterns.
2. Pagination & Projections
    - Always paginate search endpoints.
    - Use projections (only fetch necessary fields, e.g. omit large attributes blobs if not needed).
3. Bulk operations
    - Nightly job uses bulkWrite not individual insert/update calls.
4. Caching
    - In SDK, L1 in-process cache for hot reads: frequently used plans, carriers, metadata.
    - Optionally, L2 distributed cache (Redis) for “list of plans by year/tenant”.
5. Sharding (future)
    - Initially: replica set only.
    - If data grows (e.g. multi-tenant with millions of rates), shard rates collection by {tenantId, year}.

7. Data Purge / Retention

Design a retention policy up front:

1. Logical retention (business-visible)

- Keep current version and maybe N-1 historical versions available via API (e.g. for audits).
- Additional versions archived.

2. Physical purge strategy

   Option A – TTL indexes:

   - For old archived collections:
     - archive_plans, archive_rates with archivedAt field and TTL index.
     - Example: purge after 7 years:

```
db.archive_plans.createIndex({ archivedAt: 1 }, { expireAfterSeconds: 7*365*24*3600 })
```

   -
   - Use TTL only when purge is purely time-based.

3. Option B – Archiving job:
   - Periodic job:
     - Finds outdated versions (e.g. version < currentVersion - 1).
     - Moves them to cheaper storage (e.g. object storage as JSON/Parquet).
     - Deletes from primary Mongo collections.
4. Soft-delete
   - isDeleted true for logically deleted plans.
   - SDK filters them out by default unless a special includeDeleted flag is passed.

8. Monitoring & Observability

8.1 At the API layer

- Metrics (Prometheus/OpenTelemetry):
    - http_requests_total{route, status}
    - http_request_duration_seconds{route}
    - plan_search_count, plan_get_count
    - plan_api_errors_total{type}
- Logs:
    - Structured logs with correlation IDs (traceId).
    - Include tenant, route, key parameters, response status.
    - Separate logs for data access errors vs business errors.
- Tracing:
    - Distributed tracing (OpenTelemetry) across:
        - Gateway → Plan API → SDK → Mongo → Nightly Job.

8.2 At the SDK / DB layer

- Metrics:
    - db_queries_total{collection, operation}
    - db_query_latency_ms{collection, operation}
    - db_errors_total{error_type}
    - cache_hit_ratio

- Mongo monitoring:
    - Replication lag.
    - Connection pool usage.
    - Slow query logs & profiling.

8.3 At the nightly job

- Metrics:
    - job_runs_total{status}
    - job_duration_seconds
    - plans_loaded_total
    - rates_loaded_total
    - records_failed_total
- Alerts:
    - Job failure.
    - Job not run in N hours.
    - Loaded count below threshold (e.g. < 90% of previous day).

9. Security recap

- Infra: Private subnets, SGs/firewalls, no public Mongo.
- Data: Encryption at rest + TLS in transit; optional field-level encryption.
- Access:

    - Mongo users scoped per application (API vs job).
    - Principle of least privilege (read-only vs read-write).
- App:
    - Strong auth (OIDC/JWT), role-based authorization, validation.
- Ops:
    - Secret rotation.
    - Regular patching.
    - Audit logs for both APIs and DB (who changed which plan/rate)